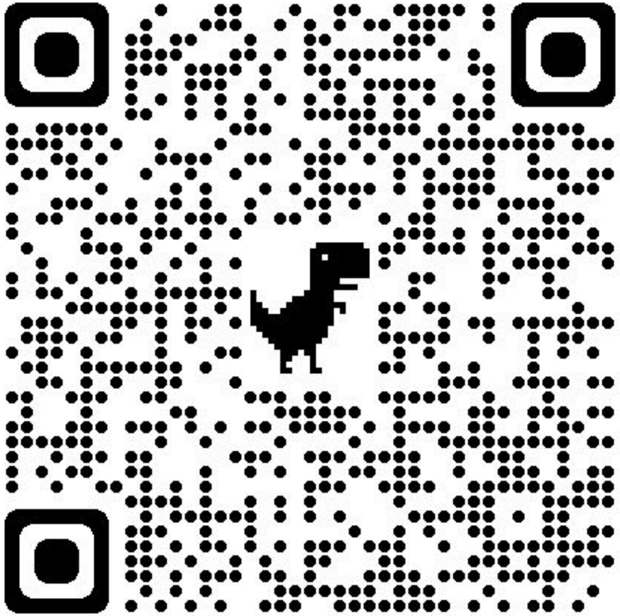
Lab Slide Link



Link

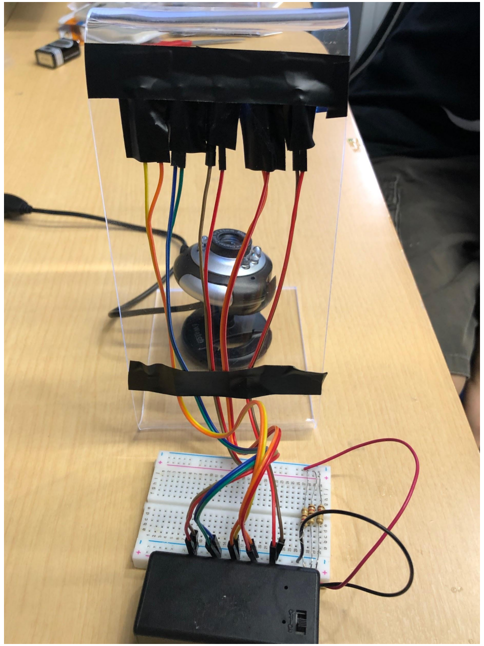
FTIR Touchpad (2)

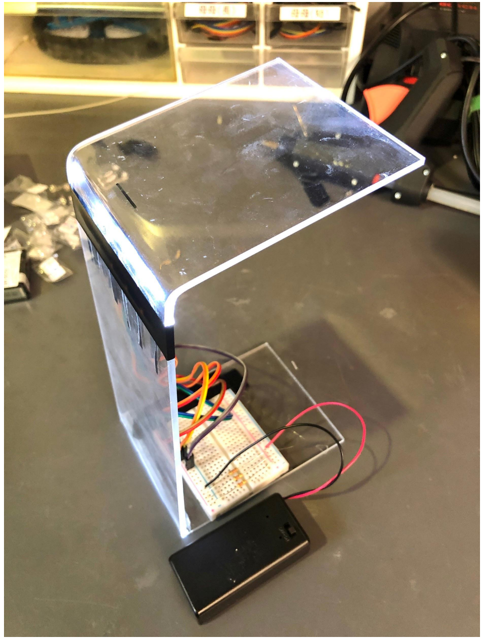
Building an acrylic multi-touch pad based on the principles of FTIR.
Week 2: Software

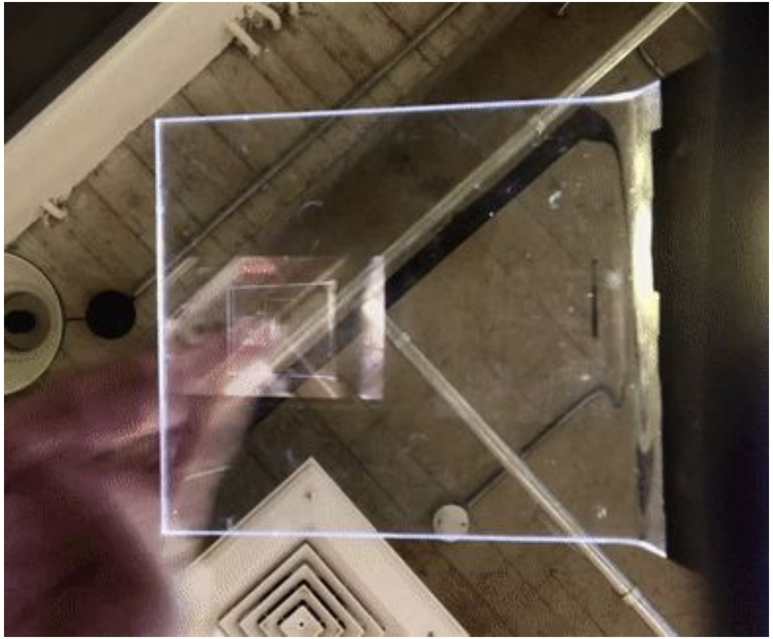
In order to build an FTIR touchpad, we need to...

- Fabricate the touchpad
- Attach LED lights and connect the circuits
- Get images from the webcam
- Know where the user is touching
- Calculate the center of the touch area

Review for last week's Lab

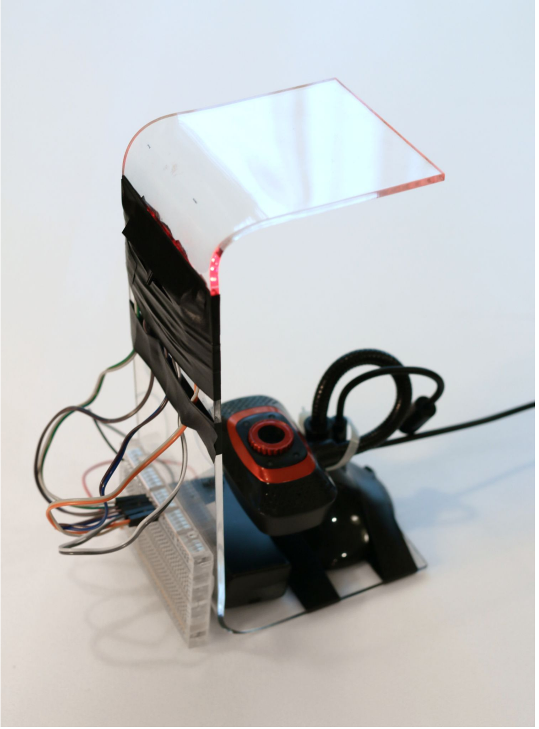






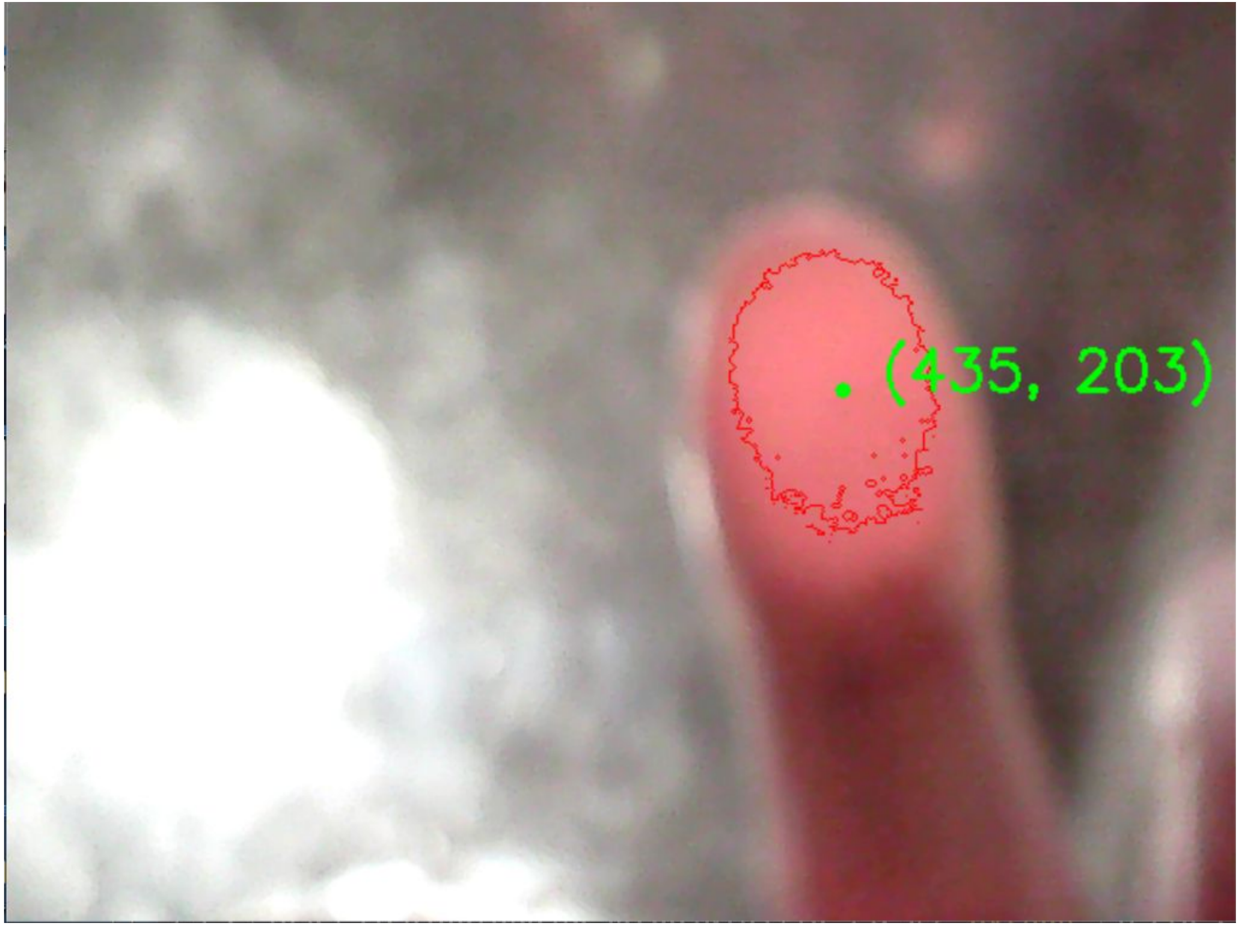
In order to build an FTIR touchpad, we need to...

- ~~Fabricate the touchpad~~
- ~~Attach LED lights and connect the circuits~~
- Get images from the webcam
- Know where the user is touching
- Calculate the center of touch area

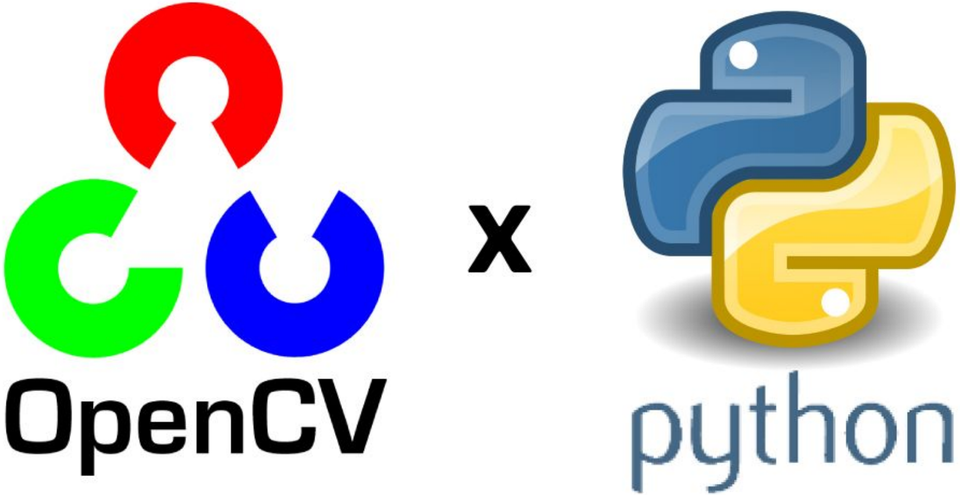


What should we do with this input?

What should we do with this input?



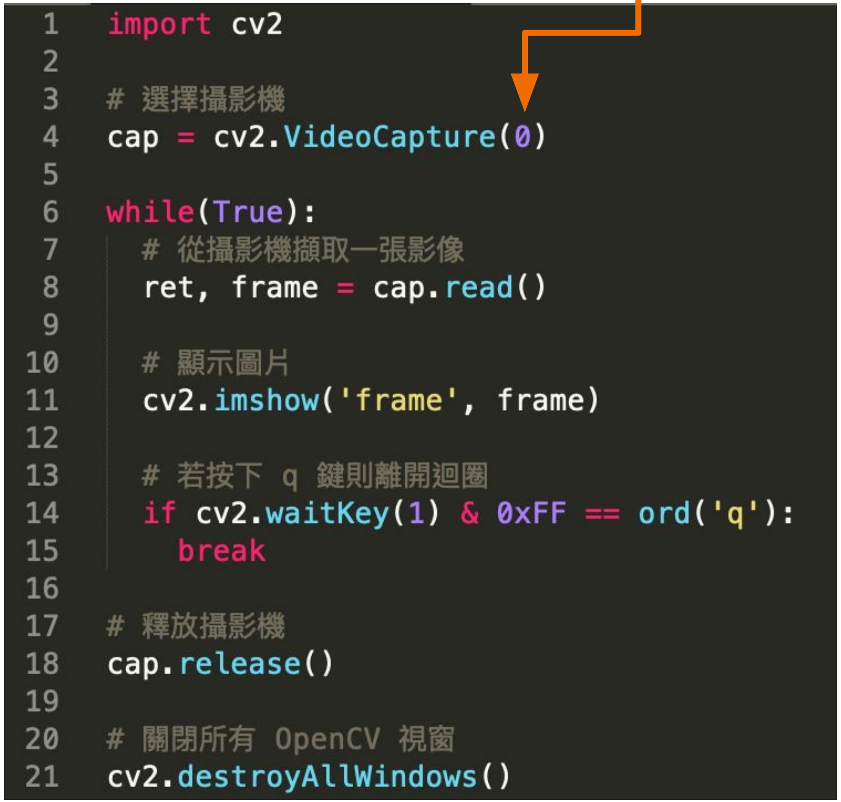
Computer Vision Library - OpenCV



Get images from the webcam

Use `cv2.VideoCapture` to instantiate a **VideoCapture** object which connects to the webcam and streams the images.

Change the number to switch between different sources



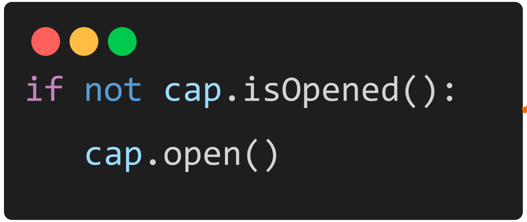
```python
import cv2

# 選擇攝影機
cap = cv2.VideoCapture(0)

while(True):
    # 從攝影機擷取一張影像
    ret, frame = cap.read()

    # 顯示圖片
    cv2.imshow('frame', frame)

    # 若按下 q 鍵則離開迴圈
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# 釋放攝影機
cap.release()

# 關閉所有 OpenCV 視窗
cv2.destroyAllWindows()
```

Get this code (ftir_video_capture.py) at [NTU COOL]

Reference:
https://blog.gtwang.org/programming/opencv-webcam-video-capture-and-file-write-tutorial/

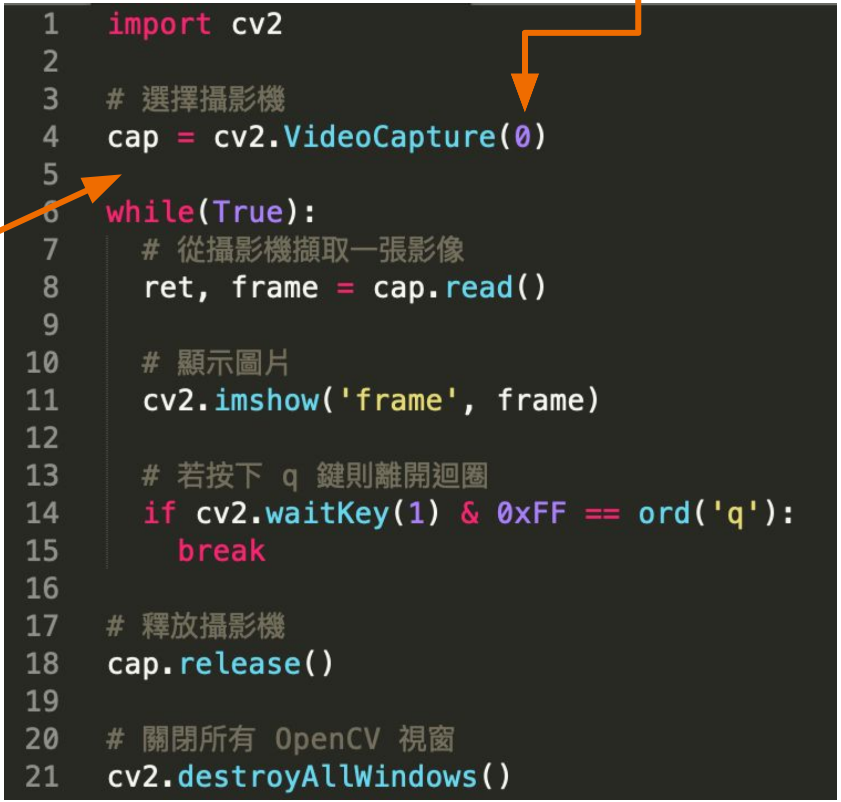
Troubleshooting

If your webcam is not opened by default, try open it manually



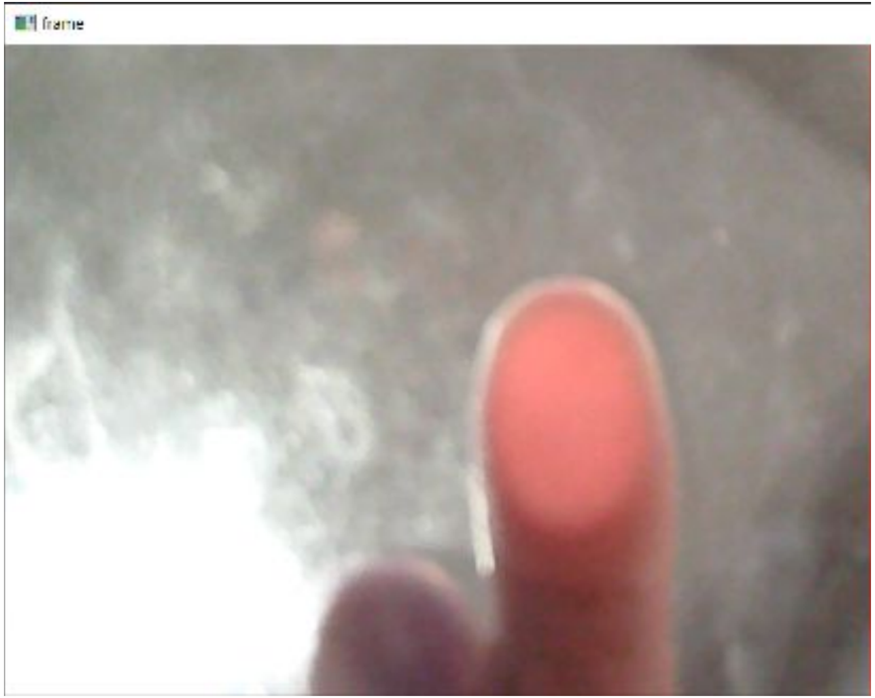
```
if not cap.isOpened():
    cap.open()
```

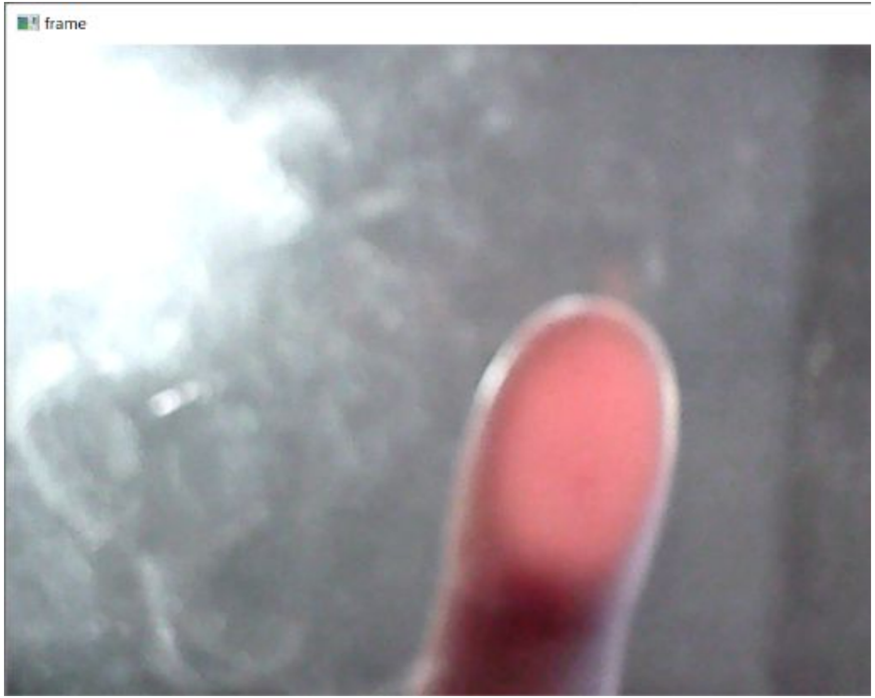
Change the number to switch between different sources



```
import cv2

# 選擇攝影機
cap = cv2.VideoCapture(0)

while(True):
  # 從攝影機擷取一張影像
  ret, frame = cap.read()

  # 顯示圖片
  cv2.imshow('frame', frame)

  # 若按下 q 鍵則離開迴圈
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# 釋放攝影機
cap.release()

# 關閉所有 OpenCV 視窗
cv2.destroyAllWindows()
```

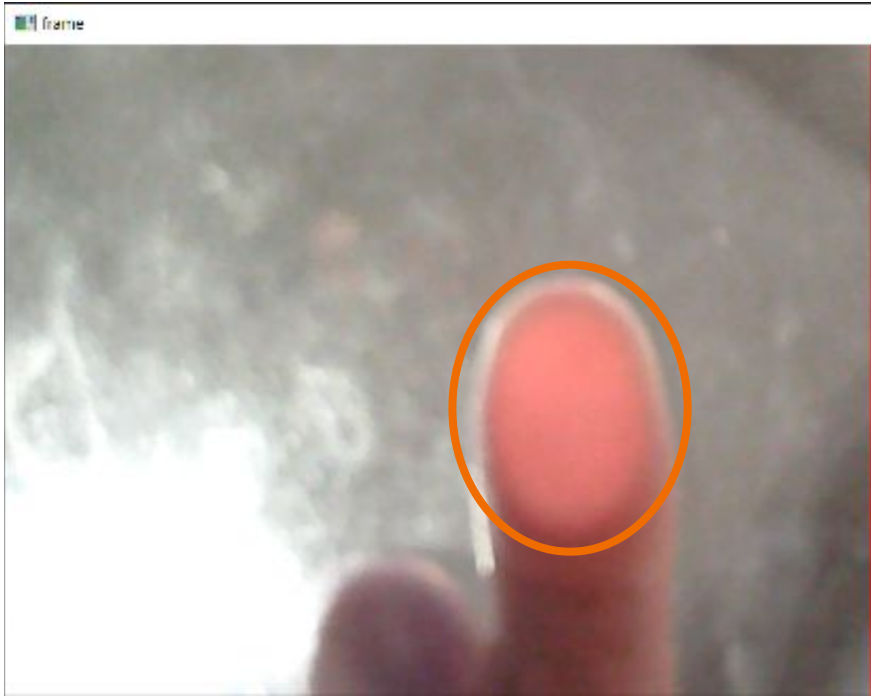
Let us see what you will see...





Let us see what you will see...

Obviously, there is a reddish spot on the image.

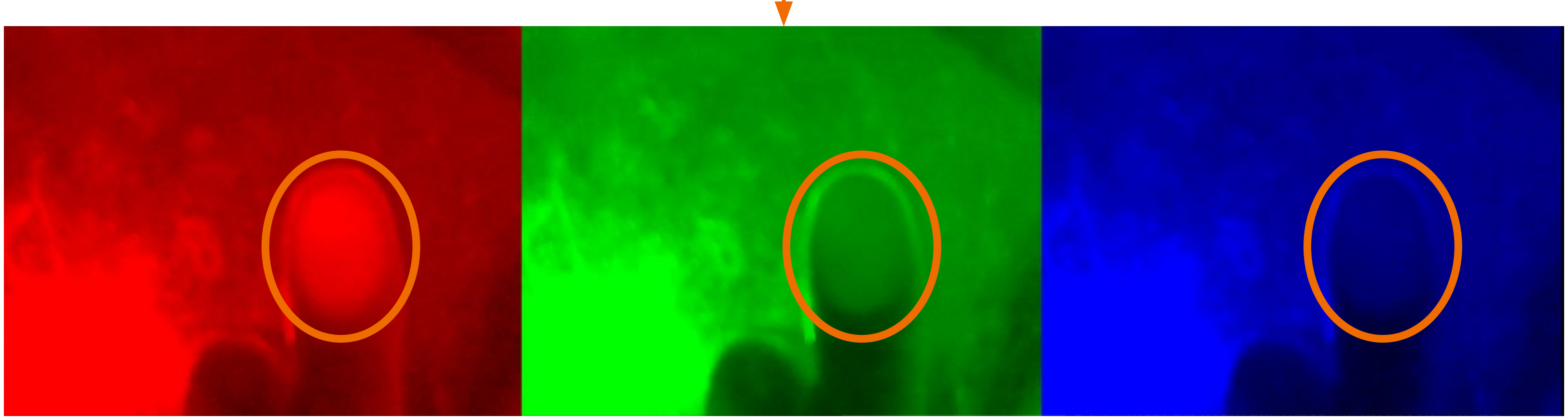


Great! We all know this is the touch point we want to locate!

Next...

Let’s split the image into separate single-channel images.

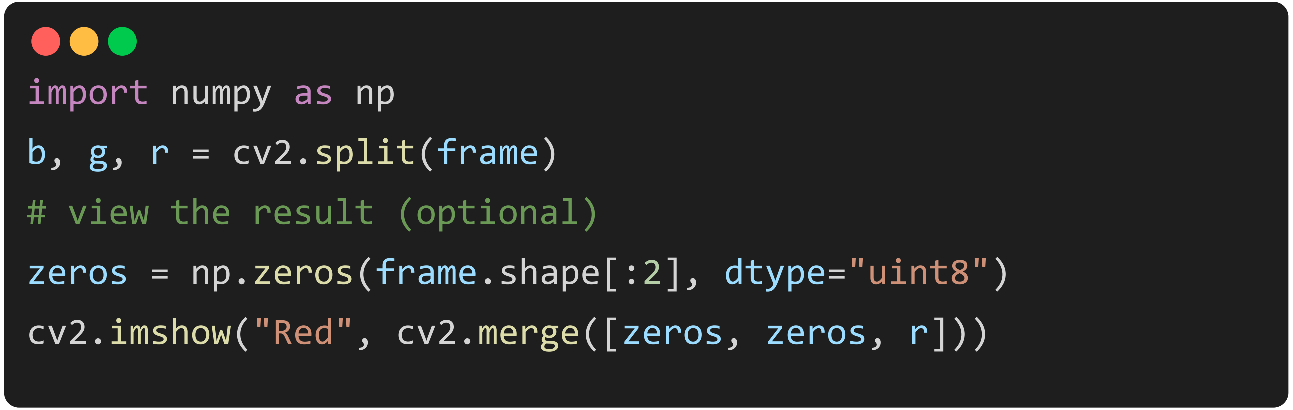
```
b, g, r = cv2.split(src)
```



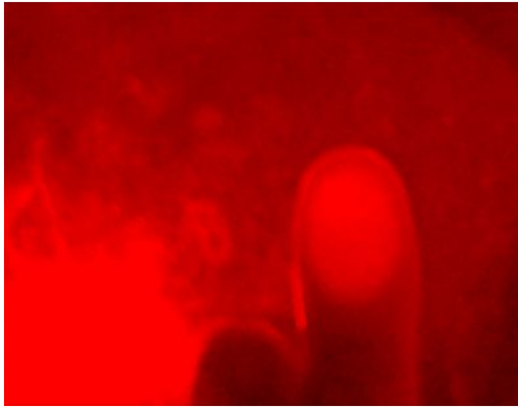
Visualization (Optional)

To view the colored images, you'll need to convert the results back to multi-channel images manually.

```
$ pip install numpy
```



```python
import numpy as np
b, g, r = cv2.split(frame)
# view the result (optional)
zeros = np.zeros(frame.shape[:2], dtype="uint8")
cv2.imshow("Red", cv2.merge([zeros, zeros, r]))
```

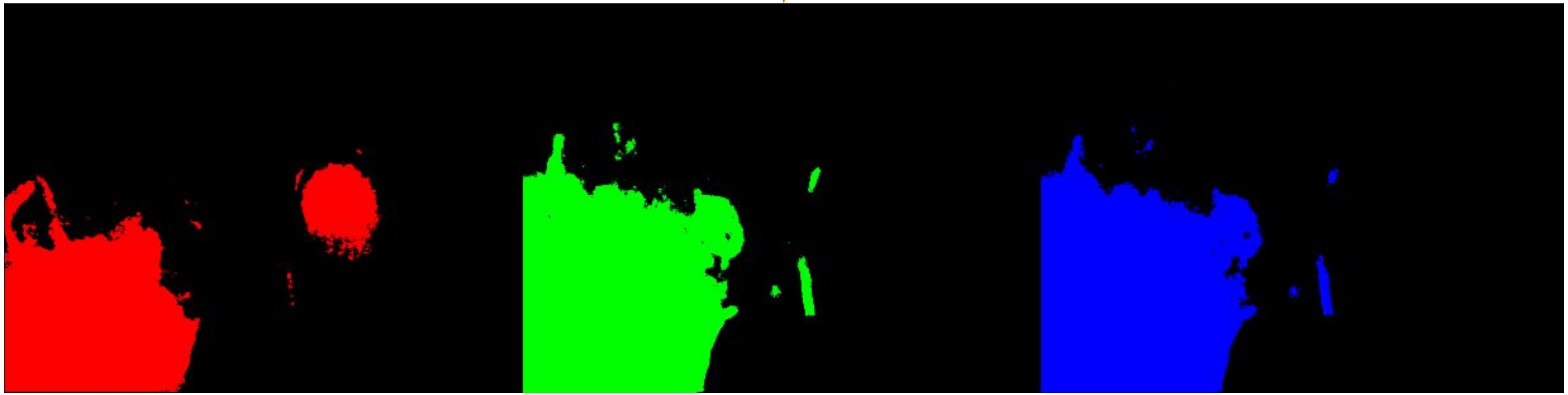


Next...

Choose an appropriate threshold (0~255) to remove the background signal.

```
_, r = cv2.threshold(r, thres, 255, cv2.THRESH_BINARY)
```

. . .

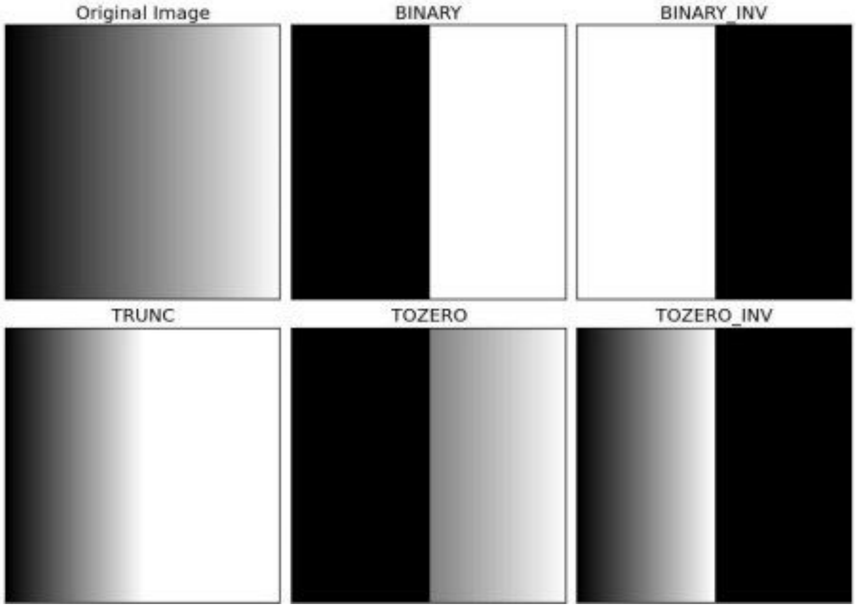


Different types of thresholds

```
_, output_img = cv2.threshold(input_img, thres, max_val, type)
```

Types:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

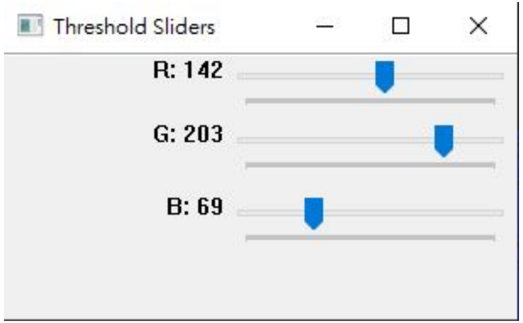


Reference: opencv-python(cv2)影象二值化函式threshold函式詳解及引數cv2.THRESH_OTSU使用

Tips (Optional)

You can tune the thresholds at runtime.

Reference: https://docs.opencv.org/4.x/d9/dc8/tutorial_py_trackbar.html

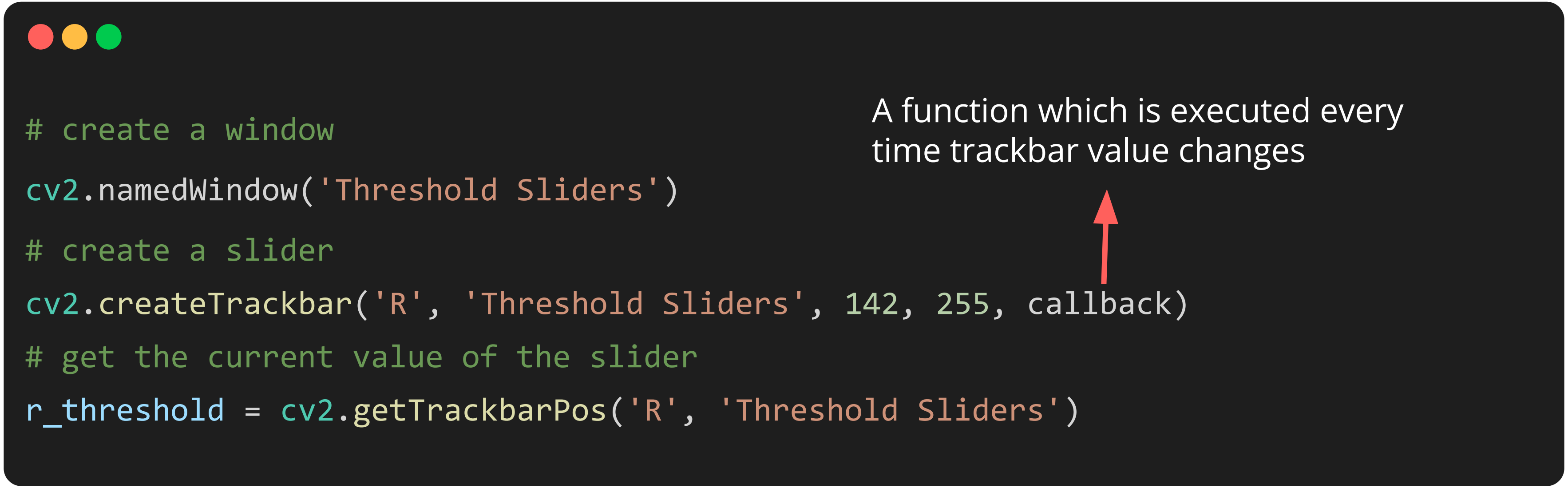


Threshold Sliders

R: 142

G: 203

B: 69

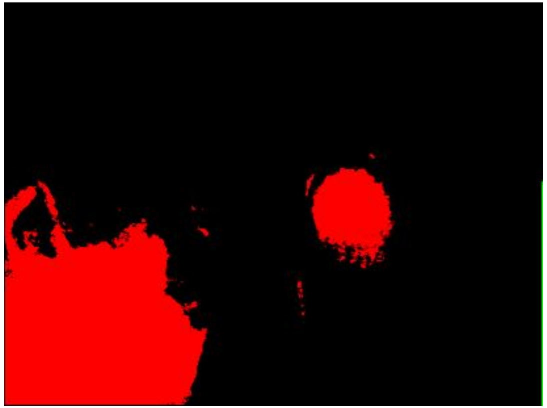


```
# create a window
cv2.namedWindow('Threshold Sliders')
# create a slider
cv2.createTrackbar('R', 'Threshold Sliders', 142, 255, callback)
# get the current value of the slider
r_threshold = cv2.getTrackbarPos('R', 'Threshold Sliders')
```

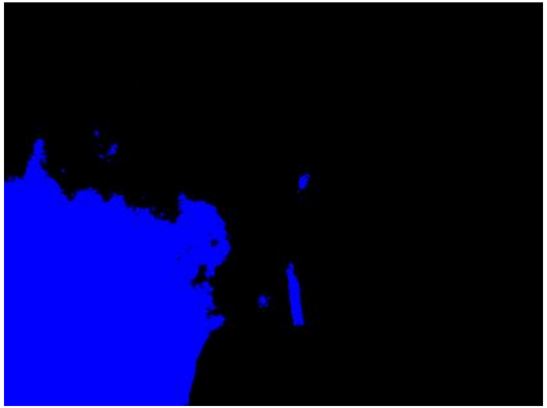
A function which is executed every time trackbar value changes

Then...

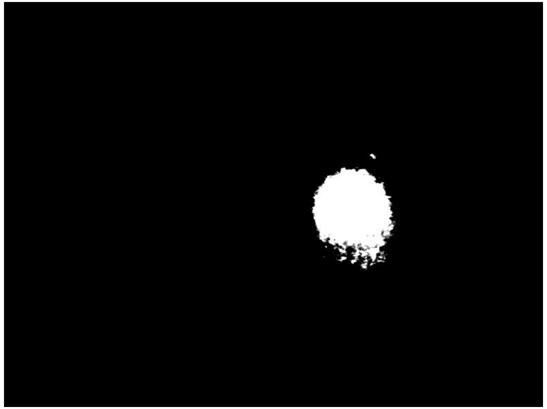
We want **Red Channel** – **Blue Channel**



−



=

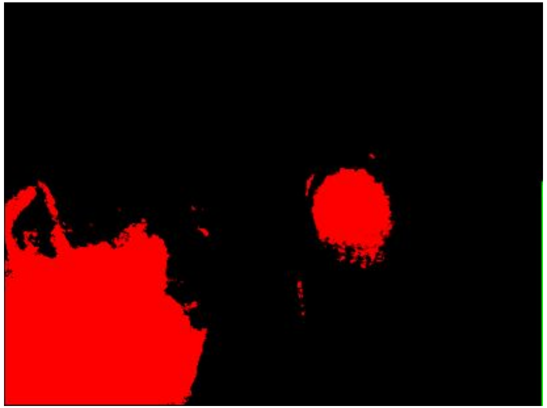


Then...

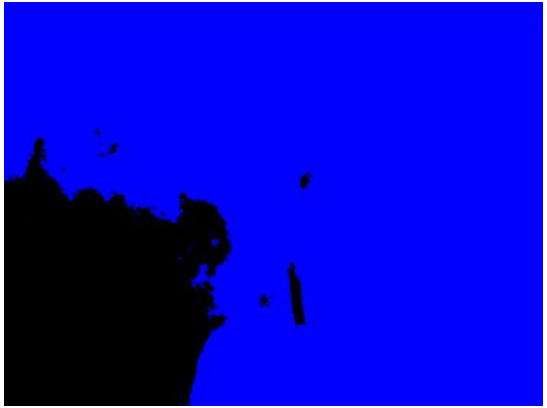
We want **Red Channel – Blue Channel**

Which can be achieved by: **Red Channel AND ¬ Blue Channel** (choose the proper flag when thresholding)

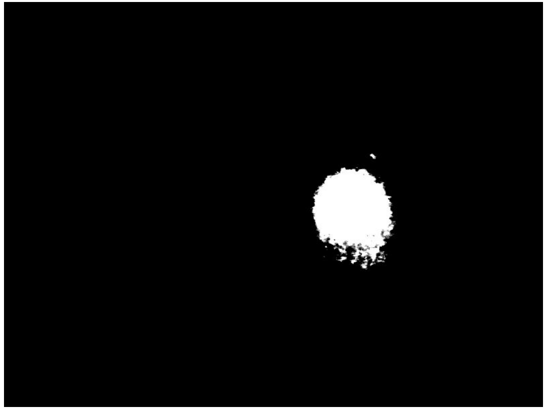
```
result = cv2.bitwise_and(r, b_inv, mask = None)
```



∧



=



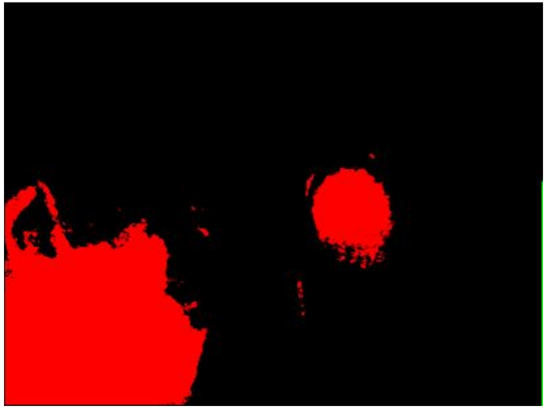
Then...

Hint: b_inv can be obtained by choosing a different flag when thresholding or performing bitwise_not to b

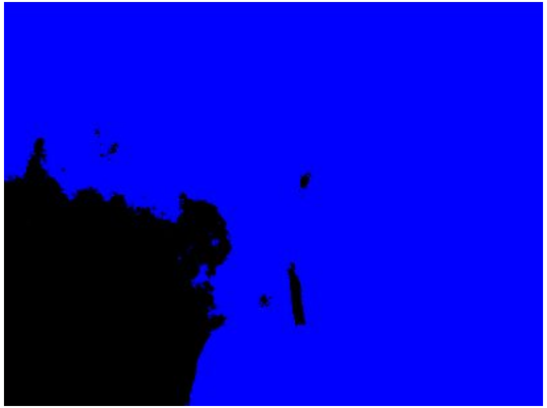
We want **Red Channel – Blue Channel**

Which can be achieved by: **Red Channel AND ¬ Blue Channel** (choose the proper flag when thresholding)

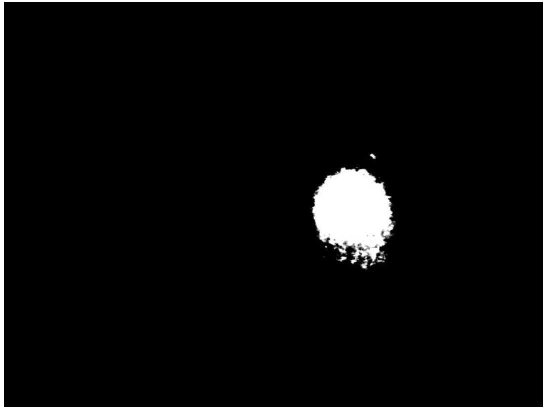
```
result = cv2.bitwise_and(r, b_inv, mask = None)
```



∧



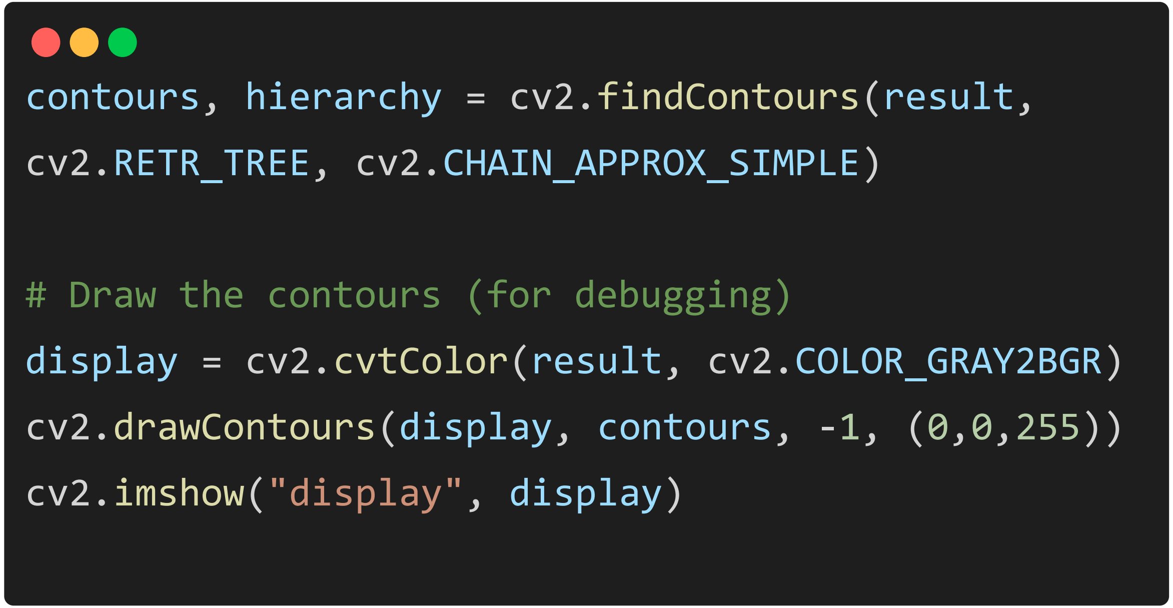
=



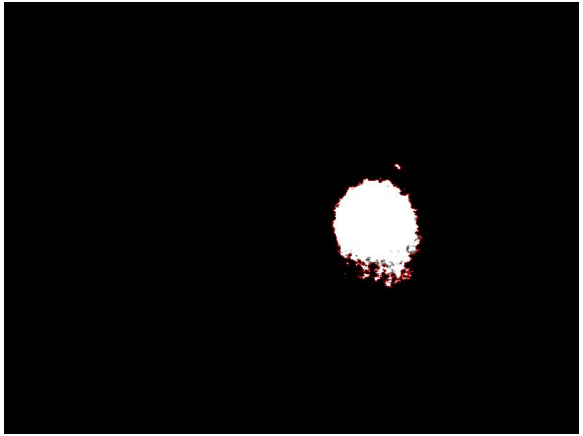
Finally...

We will extract the contours of the touched region to determine the center position.

Reference: https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html



```
contours, hierarchy = cv2.findContours(result,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours (for debugging)
display = cv2.cvtColor(result, cv2.COLOR_GRAY2BGR)
cv2.drawContours(display, contours, -1, (0,0,255))
cv2.imshow("display", display)
```

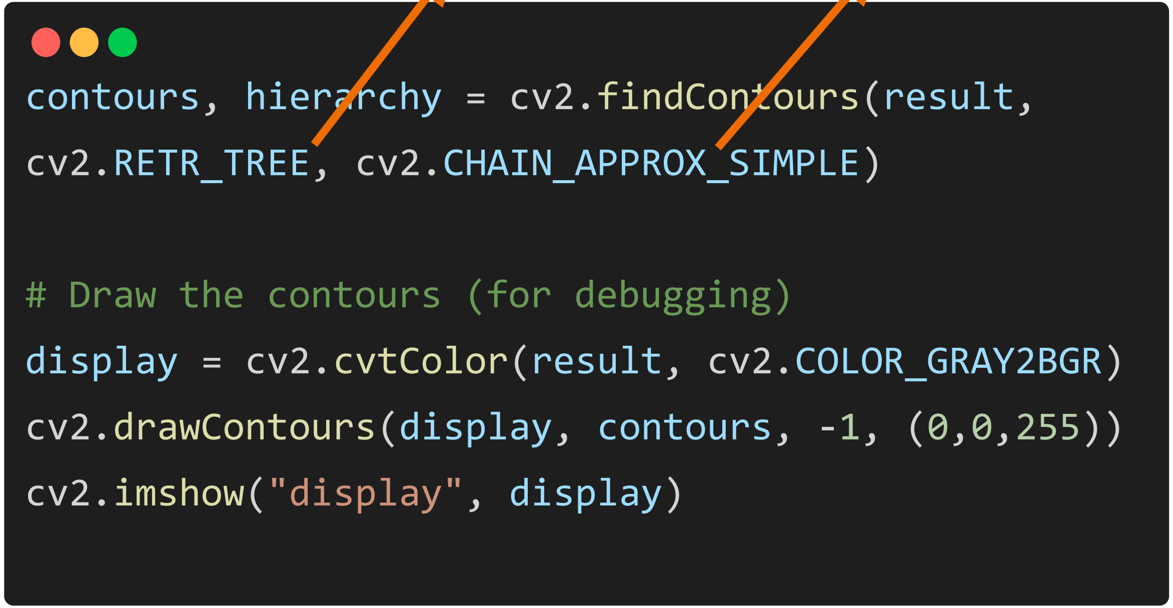


Finally...

Hint: choose a retrieval mode that best suits your detection algorithm

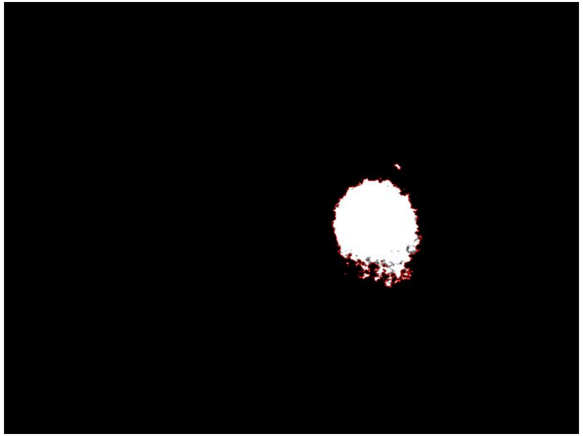
retrieval mode

approximation method



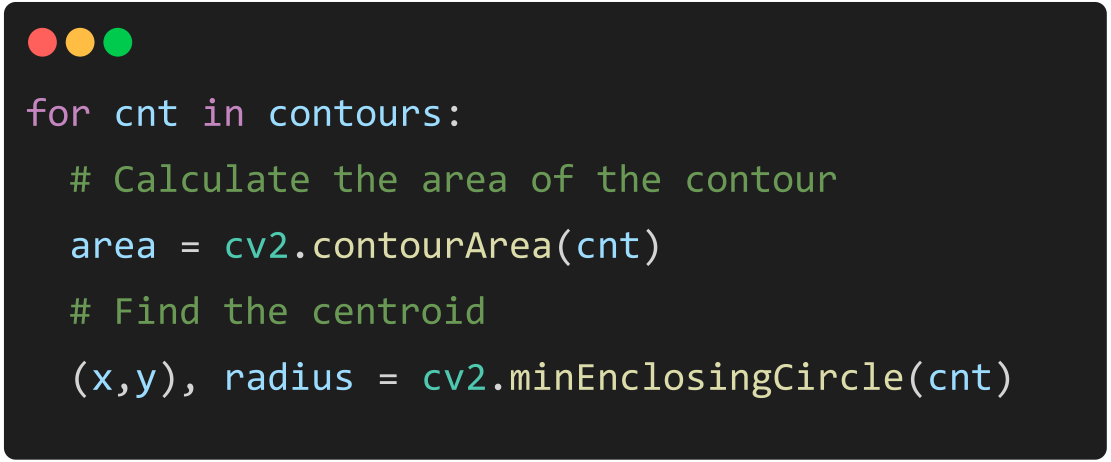
```
contours, hierarchy = cv2.findContours(result,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours (for debugging)
display = cv2.cvtColor(result, cv2.COLOR_GRAY2BGR)
cv2.drawContours(display, contours, -1, (0,0,255))
cv2.imshow("display", display)
```

Reference:
https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html



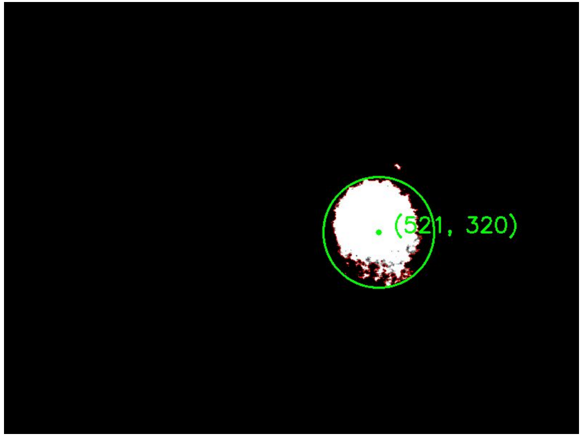
Finally...

We then examine each contour individually



```python
for cnt in contours:
    # Calculate the area of the contour
    area = cv2.contourArea(cnt)
    # Find the centroid
    (x,y), radius = cv2.minEnclosingCircle(cnt)
```

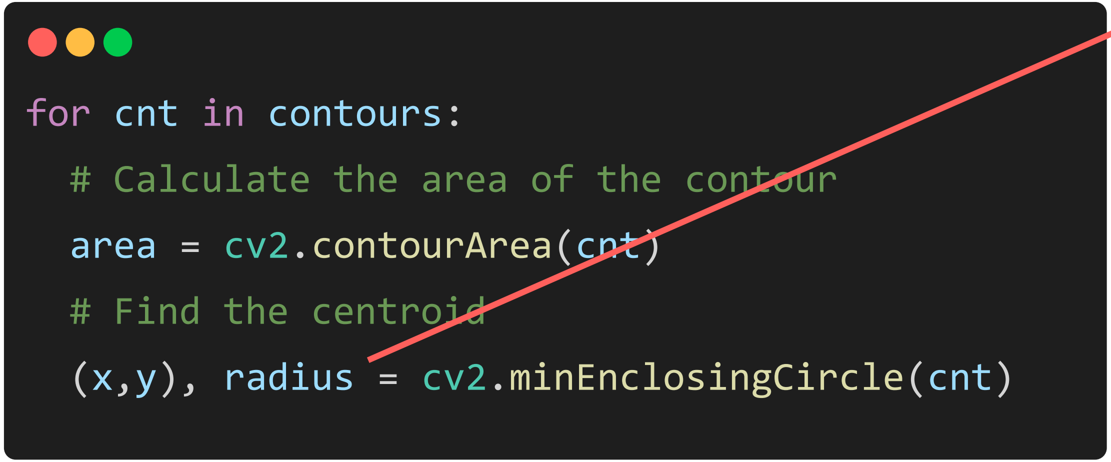
You can also find the centroid using moments



Reference:
https://docs.opencv.org/4.5.3/dd/d49/tutorial_py_contour_features.html

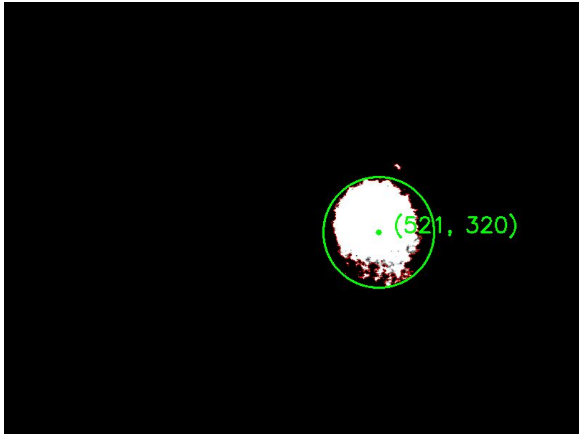
Finally...

We then examine each contour individually



```
for cnt in contours:
    # Calculate the area of the contour
    area = cv2.contourArea(cnt)
    # Find the centroid
    (x,y), radius = cv2.minEnclosingCircle(cnt)
```

Hint: the data types of the return values are not integer, you'll need to convert them manually



You can also find the centroid using moments

Reference:
https://docs.opencv.org/4.5.3/dd/d49/tutorial_py_contour_features.html

Some useful functions

```
img = cv2.flip(frame, 1)
```

```
output_img = cv2.cvtColor(input_img, FLAG)
```

- FLAG: cv2.COLOR_GRAY2BGR, cv2.COLOR_BGR2GRAY, cv2.COLOR_BGR2HSV, etc.

```
cv2.putText(img, text, pos, cv2.FONT_HERSHEY_SIMPLEX, scale, color)
```

```
cv2.circle(img, center_pos, radius, color)
```

```
M = cv2.moments(cnt)
```

Now, we can get the touch points but...

How to process these touch points over time to make them meaningful?

- **Handwritten digit recognition**
  Recognize handwritten digit number 0-9.

- **Gesture recognition (bonus)**
  Tap, Double tap, Long press, Scroll, Swipe, Zoom in/out, Rotate
  Easy Medium Hard

- **Finger ID (bonus)**
  How to know two touch points in different frame are touched by the same finger. (Recent smartphones record Finger ID up to 11)

How to implement handwritten digit recognition?

Requirements

- User writes an one-digit number (0-9) on the FTIR touchpad
- Once the user finishes writing, the system should recognize what the user wrote after 1*s*

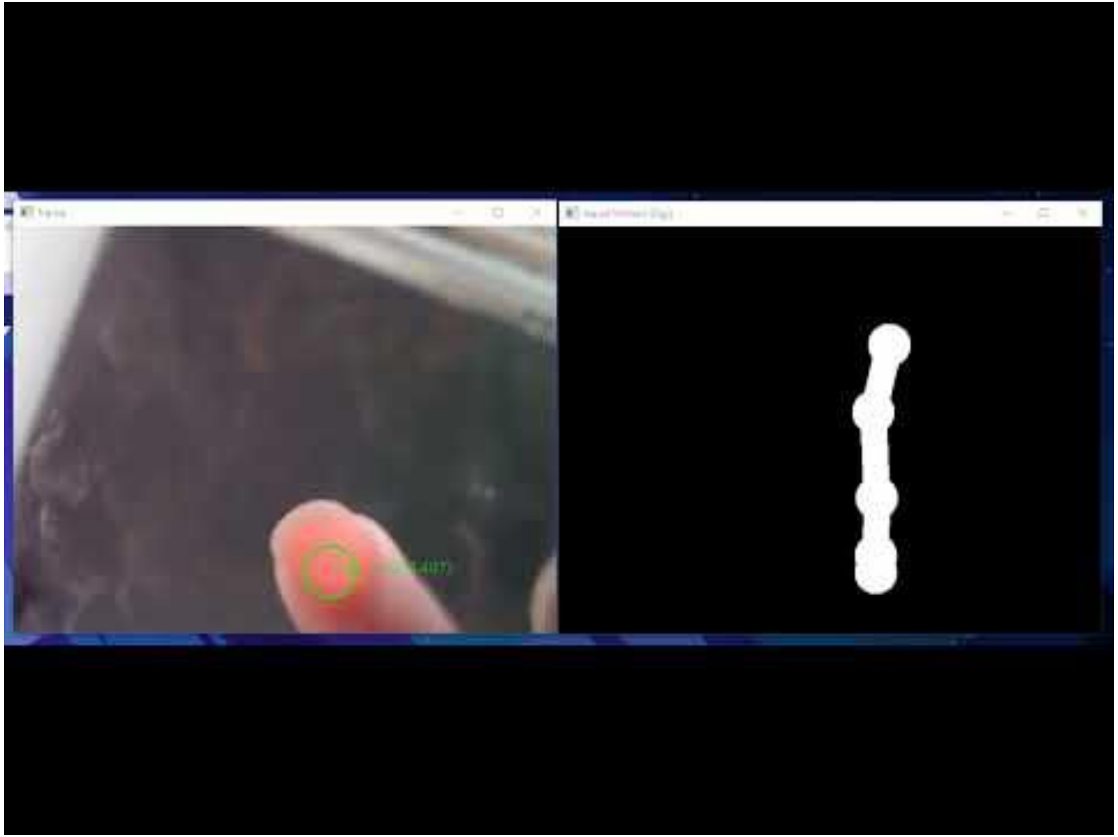
TODO

- Record the track of the touched points
- Classification (apply heuristic features or train with MNIST dataset or…)

Reference:

https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

Demo



https://youtu.be/izlrw06qJ8g

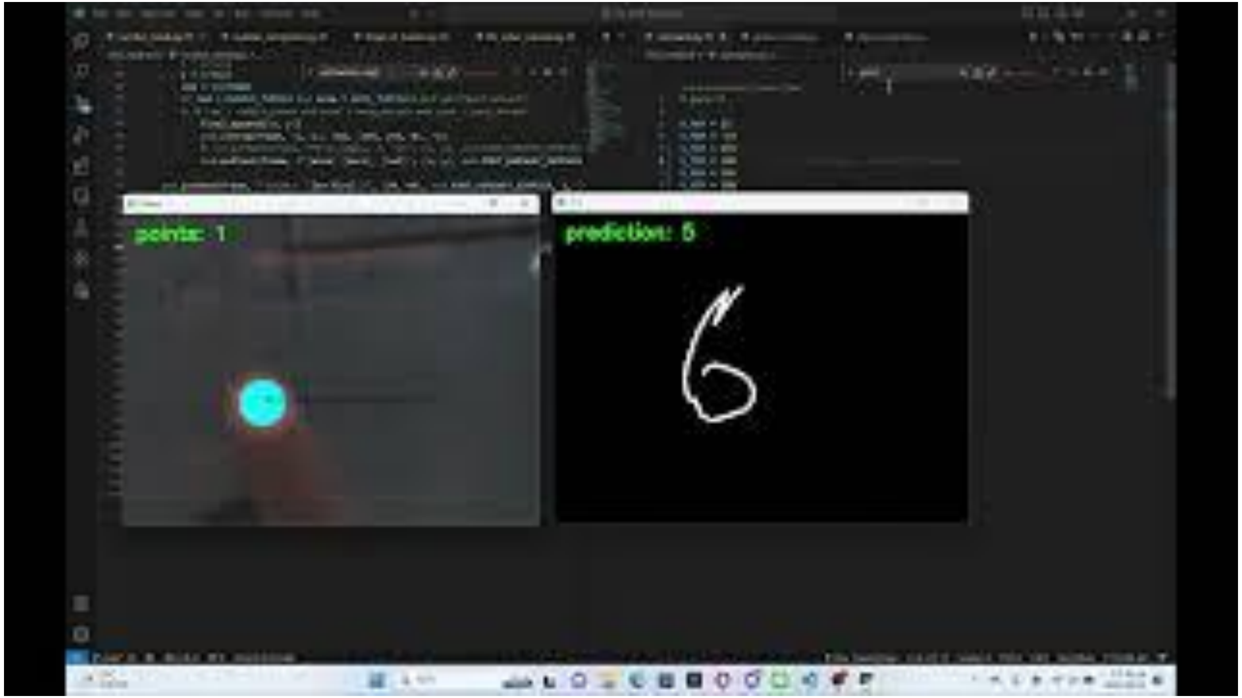
Report Requirements

- **Video** link (at most 5 min, upload to YouTube)
  - Minimum requirements
    - Show the results of **handwritten digit recognition**
  - Bonus (Please **tell us what bonus you did in the description** and **provide timestamps**)
    - Real-time digit recognition (the system predicts the results **as you write**)
    - Gesture recognition (the system should recognize the gestures **as you write**)
    - Finger ID
  - Report(You can put the video link in the report)
    - Summarize what you have learned in this lab.
    - How you can improve this device and tell us what you did.
    - Some feedback for this lab to let us know what we can improve.
    - Anything related to this lab.

Deadline: 3/19 23:59

Basic Demo

Link

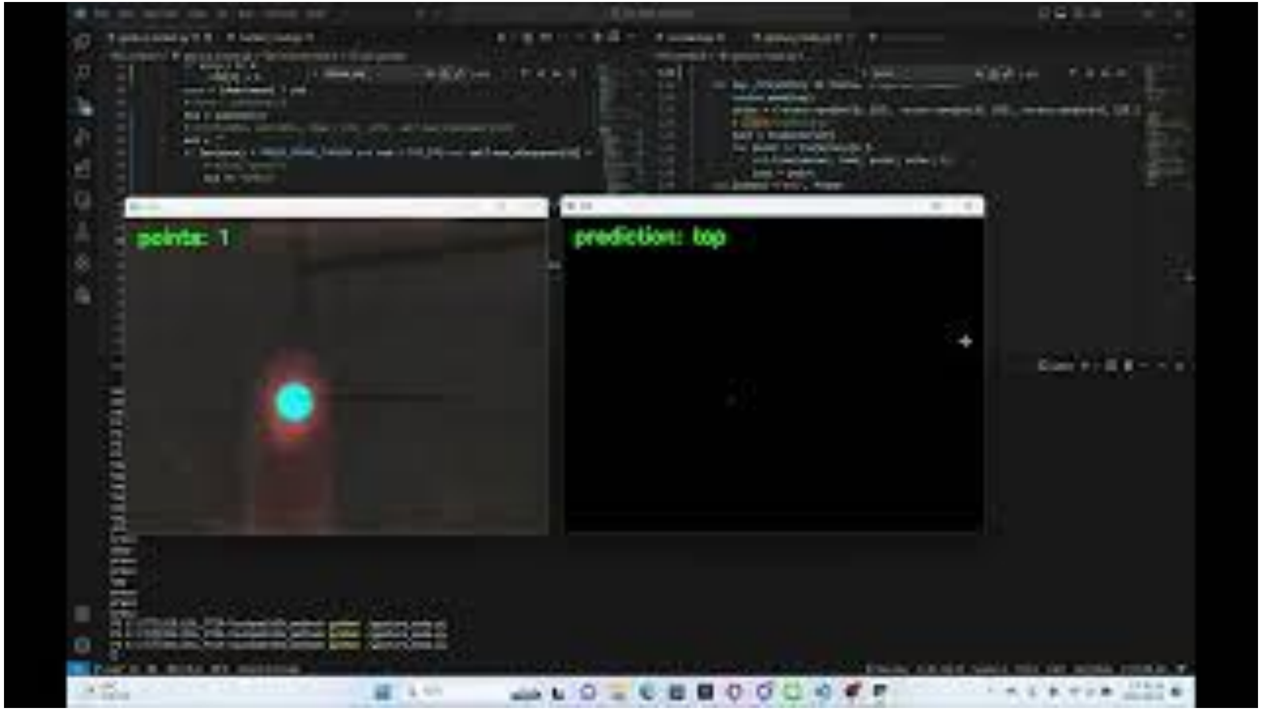


points: 1

prediction: 5

Gesture Demo

link

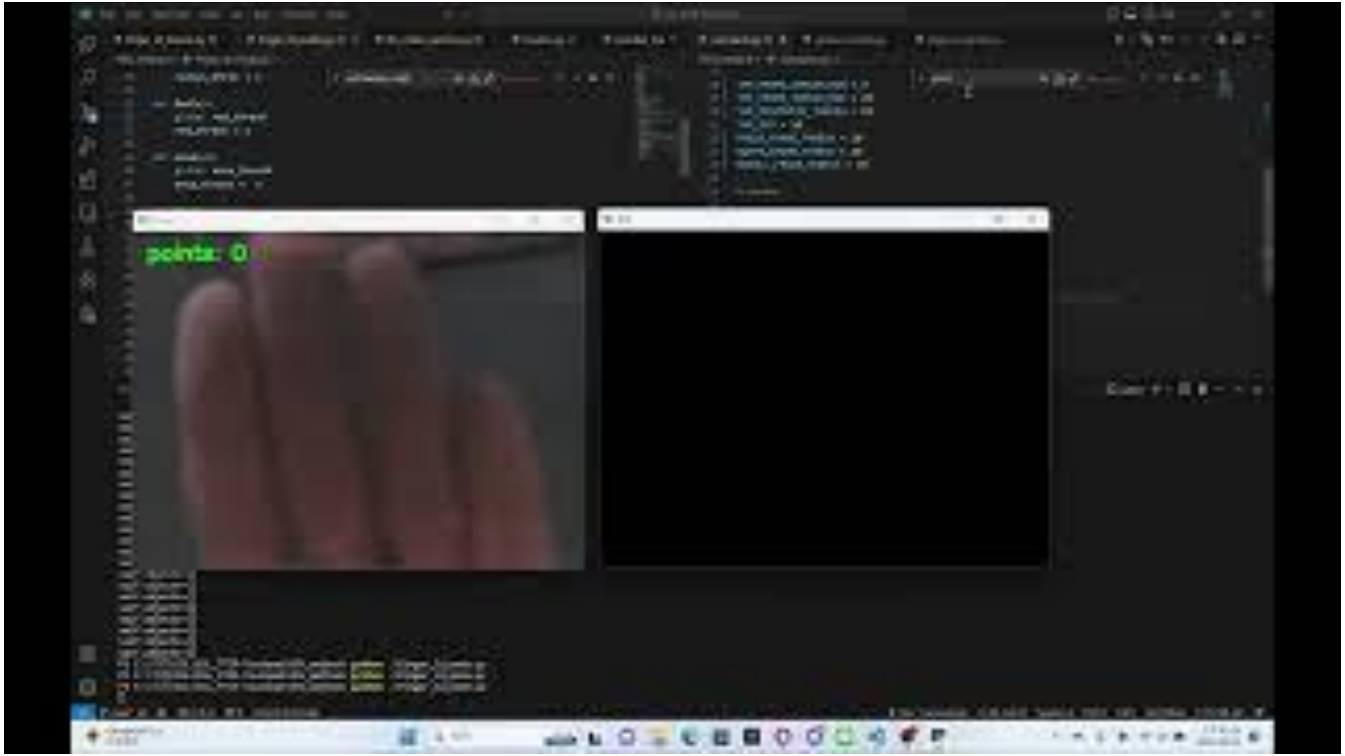


```
points: 1
```

```
prediction: top
```

Finger ID Demo

link



Any feedback is welcome🙆‍♂️

LINK